



## Question #1 (Basic Concepts)

Briefly answer each of the following questions in the space provided.

- (a) What is meant by “user mode” and “supervisor mode” in a microprocessor in the 68000 family? Why are these two modes useful to have in a microprocessor? How does the processor make transitions between these two modes?

[5 marks]

The distinction between user mode and supervisor mode is controlled by the contents of the S bit in the CPU's status register. When S=1 the CPU is said to be in the more privileged **supervisor mode**. In supervisor mode the running program can execute all of the instructions and can access all of the CPU register bits. When S=0 the CPU is said to be in the less privileged **user mode**. In user mode the running program has slightly restricted access to the CPU: (1) it cannot execute certain instructions (e.g. STOP, MOVE to SR) that could potentially have far-reaching impact on the system, (2) it cannot access the system byte of the status register, and (3) it cannot access the supervisor stack pointer (SSP). Also, (4) user mode programs cannot use the MOVEA instruction with the user stack pointer (USP) as the source or destination.

[5 marks]

Having the two privilege modes, whose different capabilities are enforced in hardware, is useful since they can form the foundation of a system architecture that can guarantee a higher degree of system security. Ordinary user programs, which are possibly less reliable or even malicious, can be required to execute in the restricted, and hence safer, user mode. Supervisor mode could be reserved for programs that can be trusted, such as initialization code, the operating system or kernel, and the input/output drivers.

[5 marks]

The processor transitions automatically from user mode to supervisor mode when exceptions (e.g. software traps or hardware interrupts) are handled by the CPU. There are two ways of transitioning from supervisor mode to user mode: (1) an instruction can be executed that clears the S bit in the status register, and (2) a return from exception (RTE) instruction can be executed when the mode before the exception was handled was user mode.

### Question #1 (Basic Concepts, cont'd)

- (b) In a multitasking system, each task is given the illusion of having exclusive use of the CPU. To maintain this illusion, a kernel has to be able to stop a task at one time, and then safely restart the same task at a later time with the CPU in exactly the same state as before. What are all of the elements of the CPU state as far as one task is concerned? Briefly explain how the CPU state (as far as one task is concerned) is saved at one time, and then restored to exactly the same state later on.

[7 marks]

As far as one task is concerned, the CPU state includes (1) the contents of all CPU registers, (2) the contents of its subroutine stack, and (3) the contents of any data structures over which it has exclusive write access. If the task does not have exclusive access to a shared data structure, then the contents of that data structure cannot be guaranteed to be unchanged in a multitasking environment.

[8 marks]

To maintain the illusion of exclusive use of the CPU, the kernel must, just before a task is restarted, restore the contents of elements (1), (2) and (3) to exactly the same state that they were in at the time when the task last executed an instruction. (1) The contents of the CPU registers are restored from copies that were saved, at the time of task pre-emption, in a data structure called the task control block (TCB). (2) The kernel does not allow the user tasks to access each other's stacks, so a task's stack will be protected while the task is suspended. (3) Data structures over which a task has exclusive access will be protected from change by the same mechanism that guarantees exclusive access.

## Question #2 (TCP/IP)

The header for a TCP segment contains fields for a source port number, a destination port number, a sequence number, an acknowledgement number, a segment checksum, plus other fields. On the other hand, the header for an IP datagram contains a source address, a destination address, an IP header checksum, plus other fields. Briefly justify the presence of these fields in the headers of the TCP segment and the IP datagram by referring to the services provided by the TCP and IP protocols, and the meaning of the fields.

[10 marks]

**TCP** provides reliable transport of a bidirectional sequence of bytes between application programs that reside in two nodes in the Internet. In the **TCP segment header**, the **source** and **destination port numbers** are included in fields because the TCP entity in the end nodes requires the port number to associate the ordered streams of data bytes with the correct application programs. The **sequence** and **acknowledge numbers** are required by the TCP entity in both of the end nodes to allow it to guarantee that all of the bytes flowing in the outbound direction reach the other node in the correct order, without any loss or duplication of data. The **segment checksum** field is used by the TCP entity in an end node to detect transmission errors in arriving segments.

[10 marks]

In the **IP datagram header**, the **source** and **destination address** must be inspected by the IP entity in each node along the datagram's path through the network to ensure that the datagram is sent correctly to the next node in the path. The **IP header checksum** is used to allow the IP header to determine whether or not the critical header information, which is being used to route the datagram, can be relied on to be error-free. If a recomputed checksum does not match the contents of the checksum field, then the datagram is discarded.

### Question #3 (Lightweight IP)

Lightweight IP is an implementation of the TCP/IP stack that was intended for use in small microcomputers. What aspects of the design of lwIP itself enhance portability across different microcontrollers? For each aspect, be sure to explain why portability is increased.

[20 marks altogether]

The portability of Lightweight IP (lwIP) is enhanced by the following design features:

- 1) [4 marks] The widely supported ANSI C programming language is used. Compilers for ANSI C are available for most CPUs.
- 2) [4 marks] lwIP is insulated from the network interface hardware by device drivers that present the same interface to lwIP. Only the device driver software needs to be changed when lwIP is moved to a new hardware environment.
- 3) [6 marks] lwIP makes very few assumptions about the operating system. It needs to have (1) tasks, (2) a message system with a nonblocking message send/post function and a blocking message receive function, and (3) programmable one-shot timer with at least 200 ms resolution (sufficient to allow TCP timeouts to be implemented). The interface to these operating system services are gathered together into an Operating System Emulation Layer. This arrangement means that any changes to accommodate a new OS are located in one easy-to-find spot.
- 4) [4 marks] Internally, lwIP uses its own buffer system, implemented in ANSI C. In this way, the efficiency of lwIP can be optimized for the purposes of TCP-UDP/IP regardless of how any native buffer system is implemented in the operating system.
- 5) [2 marks] lwIP presents to the application programs a high-level, simplified Application Programming Interface (API). This restricted API makes it more likely that application programs will port more easily to different hardware platforms along with lwIP.

#### Question #4 (Assembly Language Programming)

Write an assembly language subroutine, called FIELD\_MANIP, that receives one 32-bit pointer and one 16-bit unsigned count value on the stack. Just before FIELD\_MANIP is called, the pointer is pushed onto the stack first immediately before the count value is pushed. The pointer points to the first byte in an array of packed data bytes. The subroutine is to scan through the array and to process each byte as follows: bit #5 is to be complemented, and the values of bits #3 and #1 are to be exchanged. The two values on the stack are not to be altered by the time that the subroutine returns execution control to the calling program.

[30 marks]

```
FIELD_MANIP:
    MOVEM.L   A0/D0-D3,-(SP) /* save CPU regs */

    MOVEA.L   28(SP),A0      /* recover the array ptr */
    MOVE.W    24(SP),D0      /* recover the byte count */
    BEQ       EXIT_FM       /* exit if array empty */

LOOP_FM:
    MOVE.B    (A0),D1        /* read next data byte */

    MOVE.B    D1,D2          /* extract bit #3 */
    ANDI.B    #0b00001000,D2
    LSR.B     #2,D2          /* shift to position #1 */

    MOVE.B    D1,D3          /* extract bit #1 */
    ANDI.B    #0b00000010,D3
    LSL.B     #2,D3          /* shift to position #3 */

    BCHG.B    #5,D1          /* complement bit #5 */

    ANDI.B    #0b11110101,D1 /* clr old bits #1 & 3 */
    OR.B      D2,D1          /* OR in new bit #1 */
    OR.B      D3,D1          /* OR in new bit #3 */

    MOVE.B    D1,(A0)+      /* return byte to array */
    SUBQ.W    #1,D0         /* decrement byte count */
    BNE       LOOP_FM       /* loop if not finished */

EXIT_FM:
    MOVEM.L   (SP)+,A0/D0-D3 /* restore CPU regs */
    RTS
```